Lecture 2 – Pumping lemma, Equivalent and Minimal Representations

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2026

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 1

- Deterministic Finite Automaton (DFA): $A = (Q, \Sigma, \delta, q_0, F)$
- Extended transition function δ^*
- The language recognized by the DFA A is the language

$$L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- Languages recognized by some DFA are called regular
- Finite automata encode only finite information, but can recognize infinite languages
- Product automaton, intersection of reg. languages is regular
- Myhill–Nerode theorem (DFAs $\leftrightsquigarrow$ right congruences of Σ^* of finite index where L is a union of classes)

1.4 Pumping Lemma

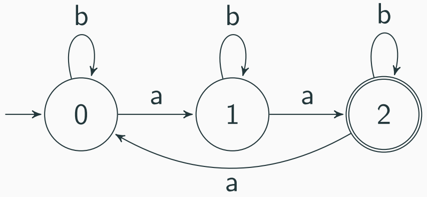
Recall: A language that is not regular

Example

$L = \{0^n 1^n \mid n \geq 0\}$ is not regular.

- Intuition: the automaton cannot ‘remember’ arbitrarily large n using only finitely many states
- Formalization: Myhill-Nerode theorem, or Pumping lemma

Illustration: a regular language



Idea: Computation as a walk on the state diagram. Is there a loop?

- $abbbba = a(b)bbba$; for all $k \geq 0$ we have $a(b)^k bbba \in L(A)$
- $aaaaba = (aaa)aba$; for all $k \geq 0$ we have $(aaa)^i aba \in L(A)$
- aa cannot be pumped but it is too short: $|aa| < n = 3$

Pumping lemma

Theorem (Pumping Lemma For Regular Languages)

Let L be a regular language. Then there exists a constant $n \in \mathbb{N}$ (which depends on L) such that for every string $w \in L$ such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

- $y \neq \epsilon$.
- $|xy| \leq n$.
- *For all $k \geq 0$, the string $xy^k z$ is also in L.*

Proof idea: The constant n is the number of states. Reading a word corresponds to a walk on the state diagram. Using the Pigeonhole principle, for long enough words we visit some state twice. The part of the walk between the first and second visit can be repeated (or skipped for $k = 0$).

Proof of the Pumping lemma

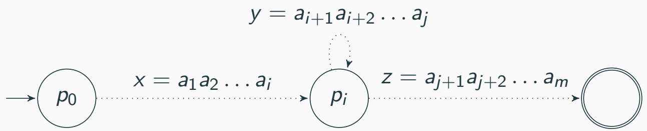
Suppose L is regular, then $L = L(A)$ for some DFA A with n states.

Take any word $w \in L$, $w = a_1 a_2 \dots a_m$ of length $m \geq n$, $a_i \in \Sigma$.

Define $\forall i \; p_i = \delta^*(q_0, a_1 a_2 \dots a_i)$. Note $p_0 = q_0$.

We have $n + 1$ p_i's and n states, therefore there are i, j such that $0 \leq i < j \leq n : p_i = p_j$.

Define: $x = a_1 a_2 \dots a_i$, $y = a_{i+1} a_{i+2} \dots a_j$, $z = a_{j+1} a_{j+2} \dots a_m$.
Note $w = xyz$.


$y = a_{i+1} a_{i+2} \dots a_j$
p_0
$x = a_1 a_2 \dots a_i$
p_i
$z = a_{j+1} a_{j+2} \dots a_m$

The loop above p_i can be repeated any number of times and the input is also accepted. $\square$

Application: proving nonregularity [an adversarial game]

Example

The language $L_{eq} = \{w; |w|_0 = |w|_1\}$ of all strings with an equal number of 0's and 1's is not a regular language.

Proof.

Suppose for contradiction that L_{eq} is regular. Take n from the pumping lemma.

- Pick $w = 0^n 1^n \in L_{eq}$.
- Break $w = xyz$ as in the pumping lemma, $y \neq \epsilon$, $|xy| \leq n$.
- Since $|xy| \leq n$ and it's at the beginning of w, it has only 0's.
- The pumping lemma says: $xz \in L_{eq}$ (for $k = 0$). However, it has less 0's and the same # of 1's as w so it's not in L_{eq}. □

More applications

Example

The language $L = \{0^i 1^i; i \geq 0\}$ is not regular. (Same proof as the previous example.)

Example

The language L_{pr} of all prime-length strings of 1's is not regular.

Proof.

Suppose it were. Take the constant n from the pumping lemma.

- Consider some prime $p \geq n + 2$, let $w = 1^p$.
- Break $w = xyz$ by the PL, let $|y| = m$. Then $|xz| = p - m$.
- By the PL, $xy^{p-m}z \in L_{pr}$. But $|xy^{p-m}z| = |xz| + (p-m)|y| = p - m + (p-m)m = (m+1)(p-m)$ which is not a prime (none of two factors are 1). $\square$

Not a characterization of regular languages!

The Pumping Lemma is not a characterization of regular languages. (It is only an implication, not an equivalence.)

Example (Nonregular language that can be 'pumped')

The language $L = \{u \in \{a, b, c\}^* \mid u = a^+ b^i c^i \text{ or } u = b^i c^j\}$ is not regular but the first symbol can be always pumped.

(a^+ means at least one a, notation from regular expressions)

Why? Use the Myhill–Nerode theorem which is a characterization or alternatively, 'Pumping Lemma with pumping near the end'.

Exercise

State and prove a pumping lemma with pumping near the end.

1.5 Equivalent and Minimal Representations

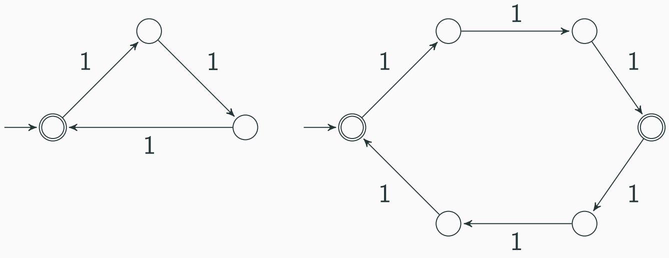
Equivalent automata

Definition

Finite automata A, B are equivalent iff they recognize the same language, that is $L(A) = L(B)$.

Example

$L = \{w \in \{1\}^* \mid |w| = 3k \text{ for some } k \geq 0\}$



Automata homomorphism

Definition

Let A_1, A_2 be DFAs. A surjective mapping $h : Q_1 \to Q_2$ is an (automata) homomorphism, if it satisfies:

- $h(\delta_1(q, x)) = \delta_2(h(q), x)$
- $h(q_{0_1}) = q_{0_2}$
- $q \in F_1 \Leftrightarrow h(q) \in F_2$

A bijective homomorphism is called an isomorphism.

(Isomorphic automata only differ by the 'names' of the states.)

Theorem (Automata Equivalence Theorem)

Let A_1, A_2 be DFAs. If there exists a homomorphism from A_1 to A_2, then A_1 and A_2 are equivalent.

NB: The converse does not hold ($A_2 \not\to A_1$ in our example)

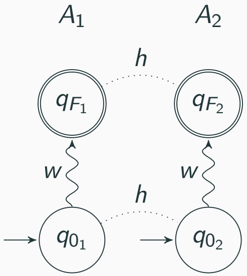
The proof: map computation to computation

For any $w \in \Sigma^*$, $q \in Q_1$, we can prove by finite induction that

$$h(\delta_1^*(q, w)) = \delta_2^*(h(q), w)$$

Then the following holds:

$$\begin{aligned} w \in L(A_1) &\Leftrightarrow \delta_1^*(q_{0_1}, w) \in F_1 \\ &\Leftrightarrow h(\delta_1^*(q_{0_1}, w)) \in F_2 \\ &\Leftrightarrow \delta_2^*(h(q_{0_1}), w) \in F_2 \\ &\Leftrightarrow \delta_2^*(q_{0_2}, w) \in F_2 \\ &\Leftrightarrow w \in L(A_2) \end{aligned}$$



$\square$

Reducing automata

The smallest DFA recognizing a given language?

Start with any DFA.

Two steps:

- remove unreachable states
- merge equivalent (indistinguishable) states

The reduced DFA is unique (up to automata isomorphism).

(Un)reachable states

Definition (Reachable states)

Let's have a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and $q \in Q$. The state q is reachable iff there exists $w \in \Sigma^*$ such that $\delta^*(q_0, w) = q$.

Algorithm (Reachable States – BFS on the state diagram)

- *set* $M_0 = \{q_0\}$
- *repeat* $M_{i+1} = M_i \cup \{q \in Q \mid (\exists p \in M_i, \exists x \in \Sigma)\ \delta(p, x) = q\}$
- *until* $M_{i+1} = M_i$
- *return* M_i

Proof of correctness and completeness.

Corectness: $M_0 \subseteq M_1 \subseteq \ldots \subseteq Q$ and consist of reachable states.
Completeness: Let q be reachable. Let $w = x_1 \ldots x_n$ be shortest such that $\delta^*(q_0, x_1 \ldots x_n) = q$. As $\delta^*(q_0, x_1 \ldots x_i) \in M_i \setminus M_{i-1}$ we get $\delta^*(q_0, x_1 \ldots x_n) = q \in M_n$. $\square$

(In)distinguishable states

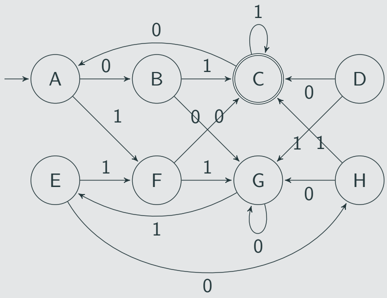
Definition (State equivalence)

States $p, q \in Q$ of a DFA A are equivalent (indistinguishable), if for all words w: $\delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$

Observation

State equivalence is indeed reflexive, symmetric and transitive.

Example



C, G distinguishable, $\delta^*(C, \epsilon) \in F$, $\delta^*(G, \epsilon) \notin F$

A,G too: $\delta^*(A, 01) = C$ accepting, $\delta^*(G, 01) = E$ not accepting.

A,E equivalent (for ϵ, $1*$ obvious, 0 goes to non-accepting, 01 & 00 meet in the same state)

Recognizing state equivalence

Distinguish accepting from nonaccepting. Then go backwards.

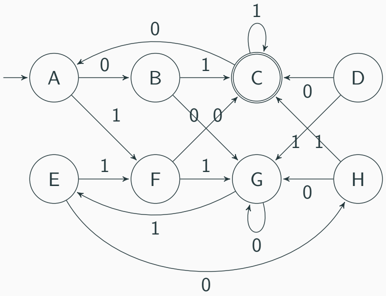
Algorithm (Finding distinguishable states in a DFA)

- *Basis: If $p \in F$ is accepting and $q \notin F$ is not, the pair $\{p, q\}$ is distinguishable.*
- *Induction: Let $p, q \in Q$ and $a \in \Sigma$. If $r = \delta(p, a)$ and $s = \delta(q, a)$ are distinguishable, then so are p and q. (Repeat until no newly distinguished pair.)*

Example 1/4

1. Accepting vs. non–accepting

B							
C	×	×					
D			×				
E			×				
F			×				
G			×				
H			×				
	A	B	C	D	E	F	G

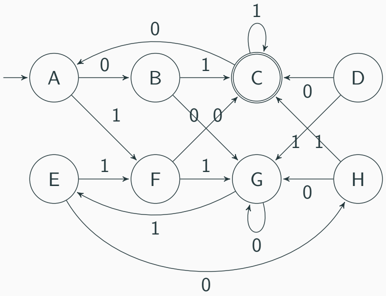


$$\begin{tikzcd}
{} \arrow[r] & A \arrow[r, "0"] \arrow[dr, "1"] & B \arrow[r, "1"] \arrow[dr, "0"] & C \arrow[ll, bend right, "0"'] \arrow[loop above, "1"] & D \arrow[l, "0"] \arrow[dl, "1"] \\
& E \arrow[r, "1"] \arrow[rrr, bend right, "0"'] & F \arrow[r, "1"] \arrow[ur, "0"] & G \arrow[ll, bend left, "1"] \arrow[loop below, "0"] & H \arrow[l, "0"] \arrow[ul, "1"]
\end{tikzcd}$$

Example 2/4

2. $\delta(q, 1) \in \mathcal{F}$ for $q \in \{B, C, H\}$

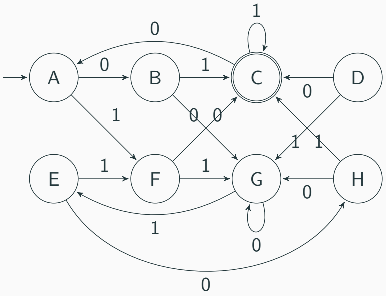
B	×						
C	×	×					
D		×	×				
E		×	×				
F		×	×				
G		×	×				
H	×		×	×	×	×	×
	A	B	C	D	E	F	G



Example 3/4

3. $\delta(q, 0) \in \mathcal{F}$ for $q \in \{D, F\}$

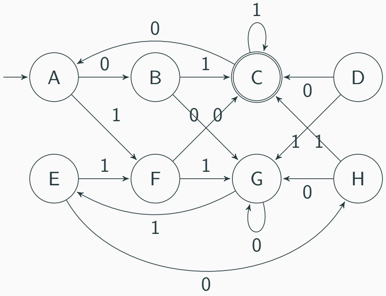
B	×						
C	×	×					
D	×	×	×				
E		×	×	×			
F	×	×	×		×		
G		×	×	×		×	
H	×		×	×	×	×	×
	A	B	C	D	E	F	G



Example 4/4

4. B and G are distinguishable, $\delta(A,0) = B$, $\delta(G,0) = G$, therefore A, G are distinguishable. Similarly, $\delta(*,0)$ for E, G goes to distinguishable states H, G.

B	×						
C	×	×					
D	×	×	×				
E		×	×	×			
F	×	×	×		×		
G	×	×	×	×	×	×	
H	×		×	×	×	×	×
	A	B	C	D	E	F	G



Equivalent pairs:

(A, E), (B, H), (D, F)

Correctness

Theorem

A pair of states is not distinguished by the algorithm, if and only if the states are equivalent.

Proof.

$\Leftarrow$ Clearly, only distinguishable pairs are distinguished.

$\Rightarrow$ Induction on the length of a shortest distinguishing word. If p, q are distinguished by $w = \epsilon$, then the algorithm distinguishes them. Now let $w = a_1 \dots a_k$. By induction, $r = \delta(p, a_1)$ and $s = \delta(q, a_1)$ are distinguished by the algorithm. But then the algorithm distinguishes p, q in the next round (following a_1-transitions backwards). $\square$

Complexity

The time complexity is polynomial in the number of states n.

- In one iteration, we consider all pairs, that is $O(n^2)$.
- In each iteration we add a cross, that means no more than $O(n^2)$ iterations.
- Together, $O(n^4)$.

The algorithm may be sped up to $O(n^2)$ by memorizing states that depend on the pair $\{r, s\}$ and following the list backwards.

Exercise

- Describe the $O(n^2)$ algorithm hinted above.
- The algorithm can also compute, for each distinguishable pair, the shortest word distinguishing that pair.

Application: testing equality of regular languages

- regular languages L, M are given by some representations
- from those construct DFA A_L, A_M recognizing L, M
- we can assume $Q_L \cap Q_L = \emptyset$ (otherwise rename the states)
- run the following algorithm:

Algorithm (Testing equivalence of automata)

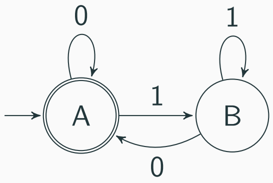
- *Construct a DFA* $B = (Q_L \cup Q_M, \Sigma, \delta_L \cup \delta_M, {q_0}_L, F_L \cup F_M)$ *as a union of states and transitions; select one (any) initial state.*
- *Test if* ${q_0}_L$ *and* ${q_0}_M$ *are equivalent. (The automata are equivalent iff their initial states are equivalent in* B*.)*

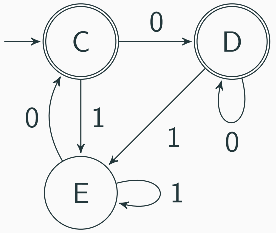
NB: Renaming states gives an isomorphic (hence equivalent) automaton. So we can always assume disjoint states. Alternatively, we can use disjoint union: $Q_B = Q_L \dot\sqcup Q_M = Q_L \times \{0\} \cup Q_M \times \{1\}$.

Example

Example

Two different DFAs recognizing $L = \{\epsilon\} \cup \{w0 \mid w \in \{0,1\}^*\}$.





B	$\times$			
C		$\times$		
D		$\times$		
E	$\times$		$\times$	$\times$
	A	B	C	D

Reduced automaton

Definition (Reduced DFA)

A DFA A is reduced iff all states are reachable and any two distinct states are distinguishable. A reduct of a DFA B is a reduced DFA equivalent to B.

Recall that δ is total. The definition implies that there is at most one fail state (a state from which no final states can be reached).

Theorem (DFA minimization)

(i) *Any two equivalent reduced automata are isomorphic.*

(ii) *Any DFA has a unique reduct (up to automata isomorphism).*

Proof.

(i) Any $q \in Q_1$ is reachable. Find a word w s.t. $q = \delta_1^*(q_{0_1}, w)$. Define $h(q) = \delta_2^*(q_{0_2}, w)$. Check that h is an isomorphism.
(ii) The construction described on the next slide works. $\square$

Algorithm: constructing the reduct

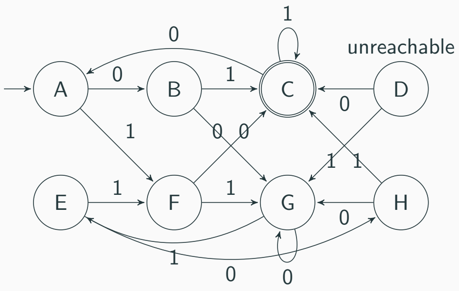
input: a DFA A

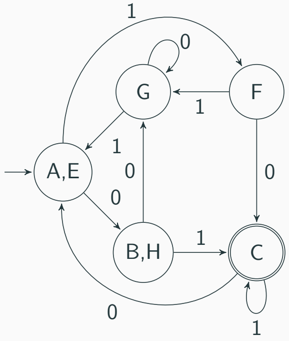
output: a DFA B which is the reduct of A

1. eliminate from A all unreachable states
2. find the indistinguishability partition on the remaining states
3. construct the reduct B:

- Q_B are the equivalence classes
- q_{0_B} is the class containing the initial state of A
- final states F_B are the classes containing some state from F_A
- the transition function: for any $a \in \Sigma$ and $S \in Q_B$ choose arbitrary $q \in S$ and define $\delta_B(S, a) = [\delta_A(q, a)]$, i.e., the class containing $\delta_A(q, a) \in Q_A$; note that this class is the same for any choice of $q \in S$ since they are all equivalent

Example





	A	B	C	E	F	G
B	×					
C	×	×				
E		×	×			
F	×	×	×	×		
G	×	×	×	×	×	
H	×		×	×	×	×

Equivalence classes:

$$\{A, E\}, \{B, H\}, \{C\}, \{F\}, \{G\}$$

Summary of Lecture 2

- Pumping lemma for regular languages (prove nonregularity)
- PL not a characterization, some nonregular can be pumped
- Equivalent automata (recognize the same language), automata homomorphism (implies automata equivalence).
- Finding reachable states: BFS on the state diagram
- Finding equivalent (indistinguishable) states: a table-filling algorithm
- Testing equivalence of DFAs, equality of regular languages
- Reduced (minimum–state) DFA, an algorithm to reduce a given DFA (using the equivalent states algorithm)